



A developer's review of LynuxWorks' BlueCat Linux SDK

Dec. 11, 2001

This article is the fourth in LinuxDevices.com's series of reviews, by software developer Jerry Epplin, of Embedded Linux toolkits. In order to gain insight into the use of each toolkit from the developer's perspective, Epplin builds Embedded Linux OS images for three different x86-based EBX form-factor single-board computers using each toolkit. Each of the toolkits is then evaluated against a common set of criteria which include ease of use, overall toolkit architecture, methods of package management, diversity of platform support, and openness of the source code. In this installment, Epplin takes a close-up look at LynuxWorks' BlueCat Linux SDK.

Toolkit strategies

When developing an Embedded Linux toolkit, a vendor can choose one of two general approaches. The more ambitious involves replacing the normal Linux development workflow with one more intuitive to the beginner and the occasional user. New users often find developing for Linux daunting, and the additional tasks necessary when deploying to an embedded system make it more so. So a toolkit vendor can attempt to encapsulate the entire Embedded Linux development process in their own environment.

This is the approach taken by all three toolkits covered in the previous articles in this series, Lineo's Embedix, MontaVista's Hard Hat Linux, and Red Hat's Embedded Linux Developer Suite (ELDS). The danger of this approach becomes evident if the vendor does not succeed in fully encapsulating the development process, requiring the user to use large portions of the underlying Linux environment anyway. An incomplete encapsulation can be worse than none at all, since it requires the user to learn the workflows of both Linux and the toolkit. As we saw in its review, Embedix is somewhat successful in this regard, as it provides a consistent environment that normally does not require the user to dip down to the underlying environment. The records of Hard Hat and ELDS are more mixed - they provide only a thin GUI veneer over Linux, requiring (to varying degrees) a more substantial understanding of the underlying Linux development processes.

The "less is more" approach to Embedded Linux toolkits

Making use of the standard Linux installation and development procedures is not necessarily a bad thing, and this suggests the second general approach to Embedded Linux toolkits. The standard Linux workflow is not unusable -- it just has chaotic documentation, and could use some utilities oriented toward deploying Linux in embedded applications. A very reasonable approach would therefore be to package Linux, add some utilities, and provide excellent documentation on the development process.

This is the approach taken by LynuxWorks with their BlueCat product. The amount of code added by LynuxWorks for this product is actually quite small; primarily consisting, as we shall see, of the BlueCat OS Loader (itself simply a Linux implementation with the ability to load another Linux system) and some fairly simple command-line utilities for constructing and deploying kernels and filesystem images. Most significantly, the entire process is extraordinarily well documented in the *User's Guide*, resulting in a complete and usable system.

The general approach of making the fewest possible modifications to the standard Linux workflow has

much to commend it. The most obvious advantage is that it requires the least effort on the part of the toolkit vendor. Start with Linux, add a few command-line tools, document it carefully and thoroughly, and you've got a product.

But the approach has other advantages . . .

- A toolkit that must track the progress of Linux chooses to modify Linux at its peril, since any modifications will have to be revisited when new versions of Linux become available. So the more it diverges from the Linux standard, the greater the resulting maintenance effort over time. This maintenance burden from incompatible versions is the usual argument used against those who predict that Linux itself will fork into numerous versions, and it holds for Embedded Linux no less than Linux in any other form.
- A simple Embedded Linux toolkit also has the advantage of appealing to current Linux users, as they need only learn the tools specific to Embedded Linux deployment.
- Finally, the approach of simple changes to Linux is more likely to add to the pool of open source code Linux code, since the utilities are likely to build on open source tools already available; as we will see, the BlueCat OS Loader is one example.

So it could be that "less is more" when either designing or shopping for an Embedded Linux toolkit.

Installing the BlueCat toolkit

BlueCat is available in Linux-hosted and Windows-hosted versions. The Windows-hosted version seems to be essentially identical to the Linux-hosted one, but running under Cygwin. One can hardly blame LynuxWorks for attempting to appeal to the larger Windows user base, and the Cygwin port could not have taken much effort to produce; but it seems obvious that the most favorable platform from which to deploy Linux-based devices is Linux. Suffice to say that this review concerns only the Linux-hosted version of BlueCat.

Host system requirements for BlueCat are refreshingly modest: '386 processor, 16MB RAM, CD-ROM drive, and 900 MB hard drive space. The documentation lists a requirement of Red Hat 6.1 or 6.2, or TurboLinux workstation 6.0; though as we will see next, BlueCat should work on almost any x86-based Linux distribution.

Installation of BlueCat is a simple process, consisting only of making a directory and running the CD-ROM based install script from that directory. The entire system is installed at this directory; and a SETUP.sh script, to be executed in a shell whenever BlueCat is to be used, is produced. Among other things, this script sets up an alternative RPM environment applying only to the BlueCat installation. That is, after SETUP.sh is run, rpm commands refer to the BlueCat package database, not to the desktop host's rpm database, if any.

This simple installation design is remarkably clever and useful. It instantly expands the range of host Linux distributions to include nearly any that can run a shell script. Thus, for example, Debian systems should be able to host BlueCat development with no effort from LynuxWorks. It also makes BlueCat admirably easy to uninstall; you've got to respect a software package having a documented uninstallation procedure consisting of "rm -rf \$BLUECAT_PREFIX". Finally, it avoids installation problems stemming from RPM incompatibilities such as those I encountered with Hard Hat. I quickly became accustomed to the BlueCat environment. I think most new users will as well, though some familiarity with the Linux command-line environment is a must.

So, BlueCat installation is simple enough that virtually nothing can go wrong.

Development Process

The development process is essentially that of Linux, with the addition of some useful utilities.

Kernel 2.2.12 is used, with some patches applied. LynuxWorks reports that substituting a later kernel should be straightforward; though they don't support it, and curiously seem to discourage it, emphasizing the "version-stabilized" kernel they provide. The x86, PowerPC, ARM/StrongARM/XScale, MIPS, and SuperH architectures are supported. Building the kernel or developing custom applications is nearly identical to the same processes for native Linux development after setting up the environment with SETUP.sh -- the normal tools are there, they just refer to the cross development tools rather than those for the native environment. The process is documented well enough for all but the very newest Linux developer; novices would do well to start with a beginner's book on native Linux development.

BlueCat provides some utilities to assist in the cross development process. 'mkernel' is a simple script to build a kernel based on a specified .config file and a location to place the resulting kernel image. Thus multiple kernel profiles are easily managed, a task that is often important when developing embedded systems.

'mkrootfs' is a powerful and flexible utility for, as its name suggests, building root file system images. You provide it a specification file defining a root file system on the target, and it builds a file system image. It can optionally scan the executables to determine which shared libraries to include, then automatically include them. mkrootfs even runs ldconfig to set up the necessary links for the libraries. mkrootfs operates from a configuration file that is similar to a shell script file. Like a shell script, the config file can invoke some Unix-like commands such as cp, rm, and mkdir to construct the target image; but the only way to control execution flow in the script is an if-else-endif construct.

BlueCat provides complete and well-designed support for target system deployment, centering around the flexible and robust utility 'mkboot'. The boot options are somewhat x86-oriented, focusing on booting in a system which has a BIOS.

To create a bootable floppy for your target, insert the floppy into the drive on your development host. Then simply run 'mkboot' with the proper flags to set up the boot sector, install the kernel, indicate where the root file system resides, and (optionally) to install the root file system on the floppy if that's where you want it. The root file system can also be on any other appropriate device on the system.

Deploying to a hard drive is similar if the disk can be temporarily mounted on the development host. In this case you simply format the disk, transfer your root file system to it, and use 'mkboot' to install the kernel and make the disk bootable. Then just move the disk to the target and boot. There's nothing fancy about this process; all of this could be done with freely available tools. But BlueCat provides helpful tools and clear documentation to help the user.

Using BlueCat OS Loader and BlueCat OS Loader Shell

One of the more useful tools provided by BlueCat is the BlueCat OS Loader, essentially a stripped-down Linux used as a bootloader. Typically you would install it on a floppy and boot it, which puts you into the BlueCat Loader Shell (BLOSH), a limited shell oriented toward installing and booting an OS over a network. From BLOSH you can specify from where to get the kernel and root file system, then boot them. A typical BLOSH sequence (taken from my test setup) might look like this:

```
> set IP 192.168.123.200
> set HOST 192.168.123.178
> set IF eth0
> set KERNEL nfs /disk2/BlueCat demo.x86/osloader/zImage.adastra
> set RFS nfs /disk2/BlueCat demo.x86/modular/modular.rfs
> boot
```

These are pretty obvious: you set your own IP address and that of the host from which you intend to get the kernel and root file system, indicate which interface to use, specify which files to get and which protocol to use in getting them; then boot the system. Besides NFS, you can use TFTP, a parallel port, or a local file system to get the kernel and root file system. You can also set kernel command-line arguments from BLOSH -- useful for using an NFS-based root file system, among other things. BLOSH commands can also be put into the file `/etc/blosh.rc` to be executed automatically on bootup, eliminating the necessity of typing in all information every time.

Most users will welcome the simplicity and clear documentation of the BlueCat OS Loader and of BLOSH. The fact that the OS Loader is itself based on Linux also has fortunate side-effects. Configuring the OS Loader for your system is just a matter of configuring the Linux kernel for your hardware; usually you'll then be able to use the same kernel for your target system.

This is exactly what I did in the above example. I configured the OS Loader for the Jumpotec-Adastra VNS-786L SBC, one of the three SBCs I'm using for this series, leaving the kernel in `zImage.adastra`. I then wrote a boot floppy using this kernel and the `osloader` root file system and booted it on the VNS-786L.

BlueCat provides a good variety of sample systems, one of which (the "modular" sample) I then loaded via BLOSH along with the previously built kernel. This process worked equally well for the other two SBCs I'm using, the WinSystems EBC-TXPlus and the Ampro Little Board/P5x.

The BlueCat OS Loader consists of the Linux kernel and BLOSH, which is run as the `init` process invoked by the kernel. Full source is provided for BLOSH, which is released under the GPL. BLOSH is extensible; you can add commands to it by following a simple and well-documented procedure.

Other observations

Notably absent from BlueCat is a tool for selecting packages to be included in the target root file system. The competing toolkits all have ways to specify what components are to be included, though with varying features and usefulness. Although this would seem to be a serious deficiency, BlueCat's lack of this type of tool is mitigated somewhat by the large number of sample systems provided; most users will find a sample close enough to their needs to serve as a basis for their own system.

Support for multiple system configurations is easily provided in BlueCat due to its simple and intuitive structure. One need only recognize that a BlueCat target consists of a kernel and a root file system, and that the kernel is defined by its `.config` file and the root file system by its `mkrootfs` configuration file. Then you can save a target configuration by simply setting aside the kernel `.config` file and the root file system configuration file. BlueCat provides a large number of sample systems, all defined in exactly this concise way.

I found working in the BlueCat environment exceptionally easy and intuitive despite (some might say because of) its command-line orientation. The tools are well thought out with intuitive naming and behavior, and backed up with documentation of the highest quality. I hope that if in the future LynuxWorks chooses to follow the rest of the market by tacking a GUI onto the front of the product, they either do it very carefully, or continue to support the command-line interface fully.

*Reprint from LinuxDevices.com
December 11, 2001*